

## COMPTE RENDU

### TP : Spring MVC Thymeleaf

Par : **Assimi DIALLO**

Encadrant : **M. YOUSSEFI**

### Spring MVC Thymeleaf

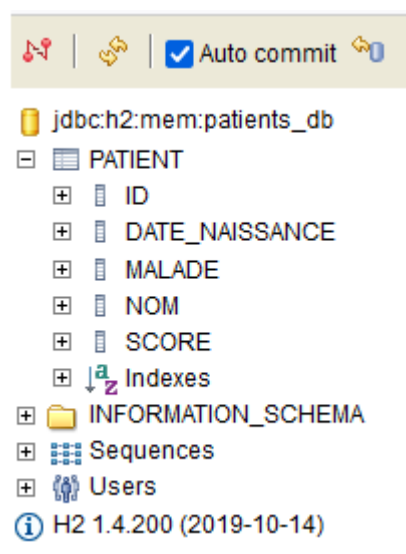
Dans ce TP nous allons faire la gestion des patients avec Spring MVC, le moteur de template Thymeleaf et Spring Data JPA.

L'application à réaliser doit afficher la liste de patients, permettre de rechercher un patient avec un mot-clé et supprimer un patient. Nous ajouterons aussi la pagination pour l'affichage de la liste des patients.

On crée donc l'entité **Patient** dans un package **entities** et grâce à Spring Data JPA nous aurons la génération de notre base de données H2.

```
Patient.java x
1 package ma.enset.patientsmvc.entities;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 import javax.persistence.*;
8 import java.util.Date;
9 @Entity
10 @Data @AllArgsConstructor @NoArgsConstructor
11 public class Patient {
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private Long id;
15     private String nom;
16     @Temporal(TemporalType.DATE)
17     private Date dateNaissance;
18     private boolean malade;
19     private int score;
20 }
21 |
```

A l'exécution on obtient sur la base de données H2 :



Par la suite on crée une interface **PatientRepository** dans un package **repositories**. Cette interface hérite de **JpaRepository** et nous permettra donc d'interagir avec la base de données.

```
PatientRepository.java x
1 package ma.enset.patientsmvc.repositories;
2
3 import ma.enset.patientsmvc.entities.Patient;
4 import org.springframework.data.domain.Page;
5 import org.springframework.data.domain.Pageable;
6 import org.springframework.data.jpa.repository.JpaRepository;
7
8 public interface PatientRepository extends JpaRepository<Patient,Long> {
9 }
10 |
```

On a sur **applications.properties** les configurations suivantes :

```
application.properties x
1 spring.datasource.url=jdbc:h2:mem:patients_db
2 spring.h2.console.enabled=true
3 server.port=8085
```

L'application va donc s'exécuter sur le port 8085.

Par la suite nous ajoutons les données dans la base de données :

```
PatientsMvcApplication.java X
1 package ma.enset.patientsmvc;
2
3 import ...
4
11
12 @SpringBootApplication
13 public class PatientsMvcApplication {
14
15     public static void main(String[] args) { SpringApplication.run(PatientsMvcApplication.class, args); }
16
17     @Bean
18     CommandLineRunner commandLineRunner(PatientRepository patientRepository){
19
20         return args->{
21             patientRepository.save(new Patient(id: null, nom: "Hassan", new Date(), malade: false, score: 12));
22             patientRepository.save(new Patient(id: null, nom: "Mohammed", new Date(), malade: true, score: 321));
23             patientRepository.save(new Patient(id: null, nom: "Yasmine", new Date(), malade: true, score: 65));
24             patientRepository.save(new Patient(id: null, nom: "Hanae", new Date(), malade: false, score: 32));
25
26             patientRepository.findAll().forEach(p->{
27                 System.out.println(p.getNom());
28             });
29         };
30     }
31
32 }
```

Quand on exécute :

localhost:8085/h2-console/login.do?jsessionId=1

Auto commit Max rows: 1000

jdbc:h2:mem:patients\_db

- PATIENT
  - ID
  - DATE\_NAISSANCE
  - MALADE
  - NOM
  - SCORE
  - Indexes
- INFORMATION\_SCHEMA
- Sequences
- Users

H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear SQL statement:

SELECT \* FROM PATIENT

SELECT \* FROM PATIENT;

ID	DATE_NAISSANCE	MALADE	NOM	SCORE
1	2022-03-27	FALSE	Hassan	12
2	2022-03-27	TRUE	Mohammed	321
3	2022-03-27	TRUE	Yasmine	65
4	2022-03-27	FALSE	Hanae	32

(4 rows, 16 ms)

Edit

Puis nous déclarons un contrôleur

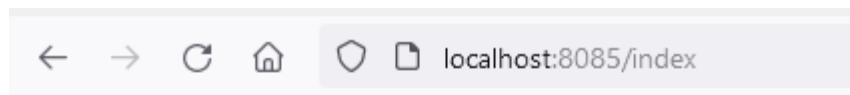
```
@Controller
@AllArgsConstructor
public class PatientController {
    private PatientRepository patientRepository;
    @GetMapping(path = "/index")
    public String patients(Model model){
        List<Patient> patients = patientRepository.findAll();
        model.addAttribute( attributeName: "listPatients",patients);
        return "patients";
    }
}
```

Le contrôleur va donc permettre la récupérer la liste de tous les patients et de la retourner dans la vue **patients** de **Thymeleaf**.

Puis on fait le traitement avec Thymeleaf en créant un fichier **patients.html** dans un dossier **templates** contenu dans le dossier **resources**. Dans **patients.html** on aura :

```
<table class="table">
    <thead>
    <tr>
        <th>ID</th><th>Nom</th><th>Date</th><th>Malade</th><th>Score</th>
    </tr>
    </thead>
    <tbody>
    <tr th:each="p:${listPatients}">
        <td th:text="${p.id}"></td>
        <td th:text="${p.nom}"></td>
        <td th:text="${p.dateNaissance}"></td>
        <td th:text="${p.malade}"></td>
        <td th:text="${p.score}"></td>
    </tr>
    </tbody>
</table>
```

L'affichage des patients se fera sur l'adresse : **localhost:8085/index**



## Liste des patients

ID	Nom	Date	Malade	Score
1	Hassan	2022-03-27	false	12
2	Mohammed	2022-03-27	true	321
3	Yasmine	2022-03-27	true	65
4	Hanae	2022-03-27	false	32

Pour améliorer l’affichage, on va ajouter bootstrap 5 :

Comme dépendance on aura :

```
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>bootstrap</artifactId>
  <version>5.1.3</version>
</dependency>
```

Puis on ajoute le lien sur le **patients.html** :

```
<title>title</title>
<link rel="stylesheet" href="/webjars/bootstrap/5.1.3/css/bootstrap.min.css">
</head>
```

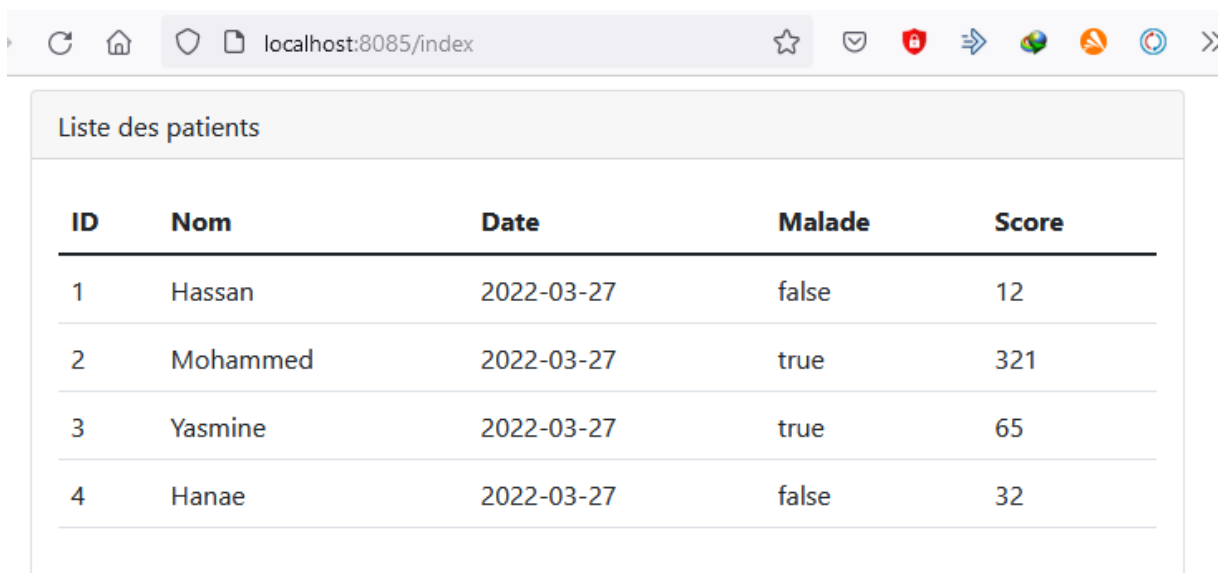
Par la suite on ajoute des classes bootstrap comme ci-dessous :

```

<div class="container mt-2">
  <div class="card">
    <div class="card-header">Liste des patients</div>
    <div class="card-body">
      <table class="table">
        <thead>
          <tr>
            <th>ID</th><th>Nom</th><th>Date</th><th>Malade</th><th>Score</th>
          </tr>
        </thead>
        <tbody>
          <tr th:each="p:${listPatients}">
            <td th:text="${p.id}"></td>
            <td th:text="${p.nom}"></td>
            <td th:text="${p.dateNaissance}"></td>
            <td th:text="${p.malade}"></td>
            <td th:text="${p.score}"></td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</div>

```

Puis quand on exécute on a :



ID	Nom	Date	Malade	Score
1	Hassan	2022-03-27	false	12
2	Mohammed	2022-03-27	true	321
3	Yasmine	2022-03-27	true	65
4	Hanae	2022-03-27	false	32

Pour pouvoir faire la pagination il nous faut plus de données. Nous allons donc passer sur une base de données MySQL de sorte qu'à chaque exécution de l'application des données soient ajoutées à la base de données :

```

<!--<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>-->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>

```

```

#spring.datasource.url=jdbc:h2:mem:patients_db
#spring.h2.console.enabled=true
server.port=8085
spring.datasource.url=jdbc:mysql://localhost:3306/patients_db?createDatabaseIfNotExist:
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MariaDBDialect
spring.jpa.show-sql=true

```

Quand on exécute l'application, on a l'ajout des données sur la base de données MySQL:

Options

					id	date_naissance	malade	nom	score
<input type="checkbox"/>	Éditer	Copier	Supprimer		1	2022-03-27	0	Hassan	12
<input type="checkbox"/>	Éditer	Copier	Supprimer		2	2022-03-27	1	Mohammed	321
<input type="checkbox"/>	Éditer	Copier	Supprimer		3	2022-03-27	1	Yasmine	65
<input type="checkbox"/>	Éditer	Copier	Supprimer		4	2022-03-27	0	Hanae	32

On remarque aussi que nous avons toujours les mêmes données sur l'adresse **localhost:8085/index**



→ ↻ 🏠 🛡️ 📄 localhost:8085/index ☆ 📧 🔒 ➡️ 🌐 🚀 🔄 >>

Liste des patients				
ID	Nom	Date	Malade	Score
1	Hassan	2022-03-27	false	12
2	Mohammed	2022-03-27	true	321
3	Yasmine	2022-03-27	true	65
4	Hanae	2022-03-27	false	32

En exécutant deux (02) autres fois, on remarque que la table s'agrandit :

↻ 🏠 🛡️ 📄 localhost:8085/index ☆ 📧 🔒 ➡️ 🌐 🚀 🔄 >

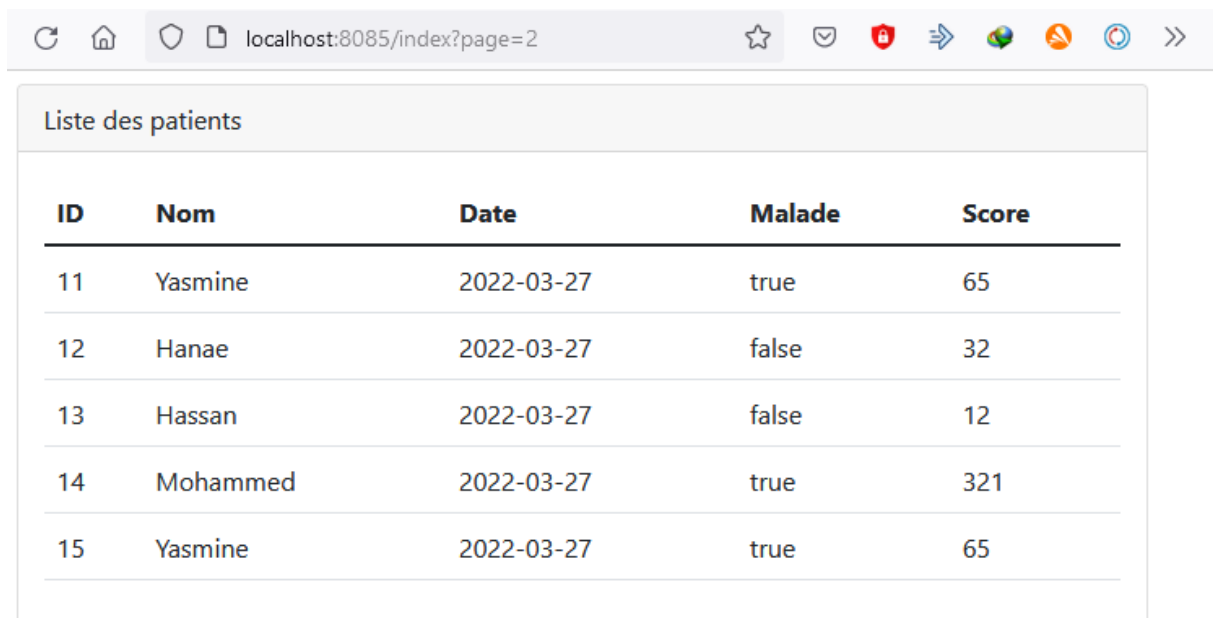
Liste des patients				
ID	Nom	Date	Malade	Score
1	Hassan	2022-03-27	false	12
2	Mohammed	2022-03-27	true	321
3	Yasmine	2022-03-27	true	65
4	Hanae	2022-03-27	false	32
5	Hassan	2022-03-27	false	12
6	Mohammed	2022-03-27	true	321
7	Yasmine	2022-03-27	true	65
8	Hanae	2022-03-27	false	32
9	Hassan	2022-03-27	false	12
10	Mohammed	2022-03-27	true	321
11	Yasmine	2022-03-27	true	65
12	Hanae	2022-03-27	false	32

Maintenant que nous avons suffisamment de données, il devient intéressant de faire la pagination :

On aura donc dans le contrôleur :

```
@Controller
@AllArgsConstructor
public class PatientController {
    private PatientRepository patientRepository;
    @GetMapping(path = "/index")
    public String patients(Model model,
        @RequestParam(name = "page",defaultValue = "0") int page,
        @RequestParam(name = "size",defaultValue = "5") int size){
        Page<Patient> pagePatients = patientRepository.findAll(PageRequest.of(page,size));
        model.addAttribute( attributeName: "listPatients",pagePatients.getContent());
        return "patients";
    }
}
```

Quand on exécute en précisant **page=2** dans l'url on remarque que la table commence à **id =11**, donc on est bien sur la **page 2** étant donné que chaque page contient 5 patients :



ID	Nom	Date	Malade	Score
11	Yasmine	2022-03-27	true	65
12	Hanae	2022-03-27	false	32
13	Hassan	2022-03-27	false	12
14	Mohammed	2022-03-27	true	321
15	Yasmine	2022-03-27	true	65

Nous allons maintenant ajouter les numéros de pages juste après la table.

On aura dans le contrôleur :

```

@Controller
@AllArgsConstructor
public class PatientController {
    private PatientRepository patientRepository;
    @GetMapping(path = "/index")
    public String patients(Model model,
        @RequestParam(name = "page",defaultValue = "0") int page,
        @RequestParam(name = "size",defaultValue = "5") int size){
        Page<Patient> pagePatients = patientRepository.findAll(PageRequest.of(page,size));
        model.addAttribute( attributeName: "listPatients",pagePatients.getContent());
        model.addAttribute( attributeName: "pages",new int[pagePatients.getTotalPages()]);
        return "patients";
    }
}

```

Et sur le fichier **patients.html** :

```

</tbody>
</table>
<ul>
    <li th:each="page,status:${pages}">
        <a th:text="${status.index}"></a>

    </li>

```

A l'exécution on a :

localhost:8085/index?page=2

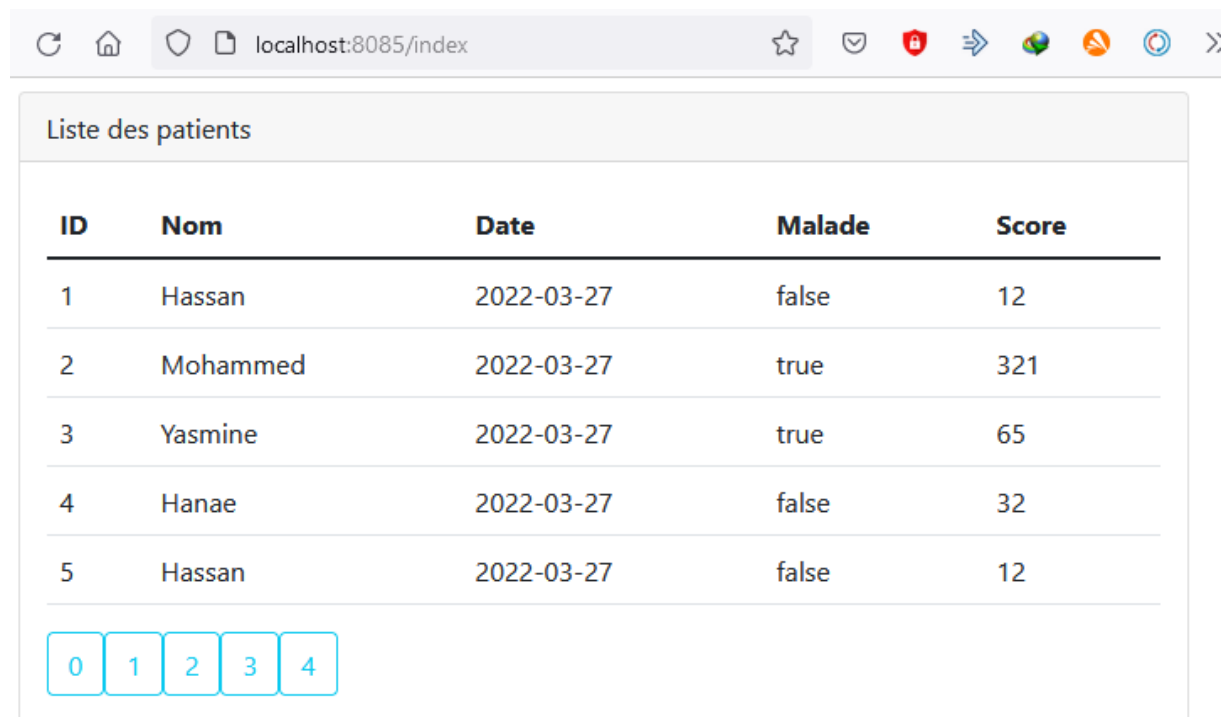
Liste des patients				
ID	Nom	Date	Malade	Score
11	Yasmine	2022-03-27	true	65
12	Hanae	2022-03-27	false	32
13	Hassan	2022-03-27	false	12
14	Mohammed	2022-03-27	true	321
15	Yasmine	2022-03-27	true	65

- 0
- 1
- 2
- 3

On arrange un peu plus l'affichage en faisant :

```
</tbody>
</table>
<ul class="nav nav-pills">
  <li th:each="page,status:${pages}">
    <a class="btn btn-outline-info" th:text="${status.index}"></a>
  </li>
</ul>
```

Et on aura :



The screenshot shows a web browser at localhost:8085/index. The page displays a table titled "Liste des patients" with 5 rows of patient data. Below the table are five pagination buttons labeled 0, 1, 2, 3, and 4. The button labeled '1' is highlighted with a blue border, indicating the current page.

ID	Nom	Date	Malade	Score
1	Hassan	2022-03-27	false	12
2	Mohammed	2022-03-27	true	321
3	Yasmine	2022-03-27	true	65
4	Hanae	2022-03-27	false	32
5	Hassan	2022-03-27	false	12

0 1 2 3 4

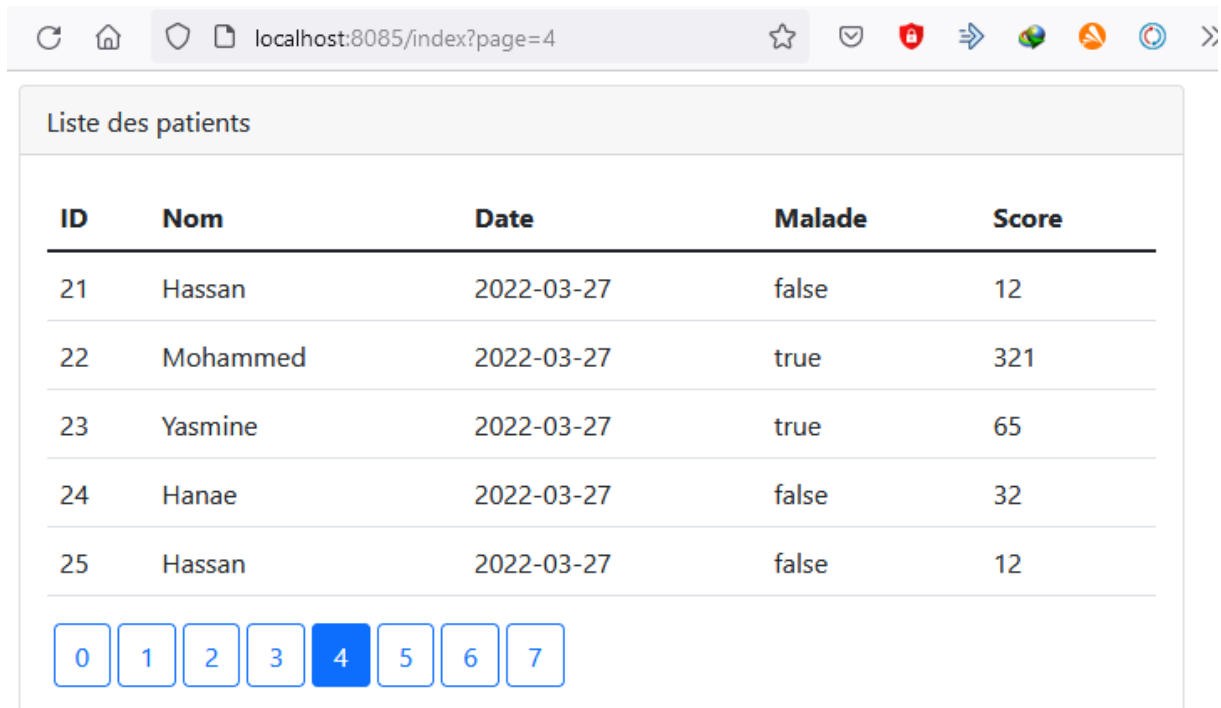
On voudra maintenant faire en sorte que quand l'utilisateur est sur une page donnée le numéro de cette page soit dans un carré bleu et ceci pour indiquer sur quelle page l'utilisateur est. On ajoute donc la page courante comme attribut du modèle comme suit :

```
public String patients(Model model,
    @RequestParam(name = "page",defaultValue = "0") int page,
    @RequestParam(name = "size",defaultValue = "5") int size){
    Page<Patient> pagePatients = patientRepository.findAll(PageRequest.of(page,size));
    model.addAttribute( attributeName: "listPatients",pagePatients.getContent());
    model.addAttribute( attributeName: "pages",new int[pagePatients.getTotalPages()]);
    model.addAttribute( attributeName: "currentPage",page);
    return "patients";
}
```

Sur le fichier patients.html on a :

```
</tbody>
</table>
<ul class="nav nav-pills">
  <li th:each="page,status:${pages}">
    <a th:class="${status.index==currentPage?'btn btn-primary ms-1':'btn btn-outline-primary ms-1}'"
      th:text="${status.index}"
      th:href="@{index(page=${status.index})}"
    ></a>
  </li>
</ul>
```

En exécutant on peut voir. Ici comme le montre l'url on est sur la page 4 et on peut le remarquer aussi sur les numéros de pages :



The screenshot shows a web browser at the URL `localhost:8085/index?page=4`. The page displays a table titled "Liste des patients" with 5 columns: ID, Nom, Date, Malade, and Score. The table contains 5 rows of patient data. Below the table is a pagination bar with buttons for pages 0 through 7. The button for page 4 is highlighted in blue, indicating the current page.

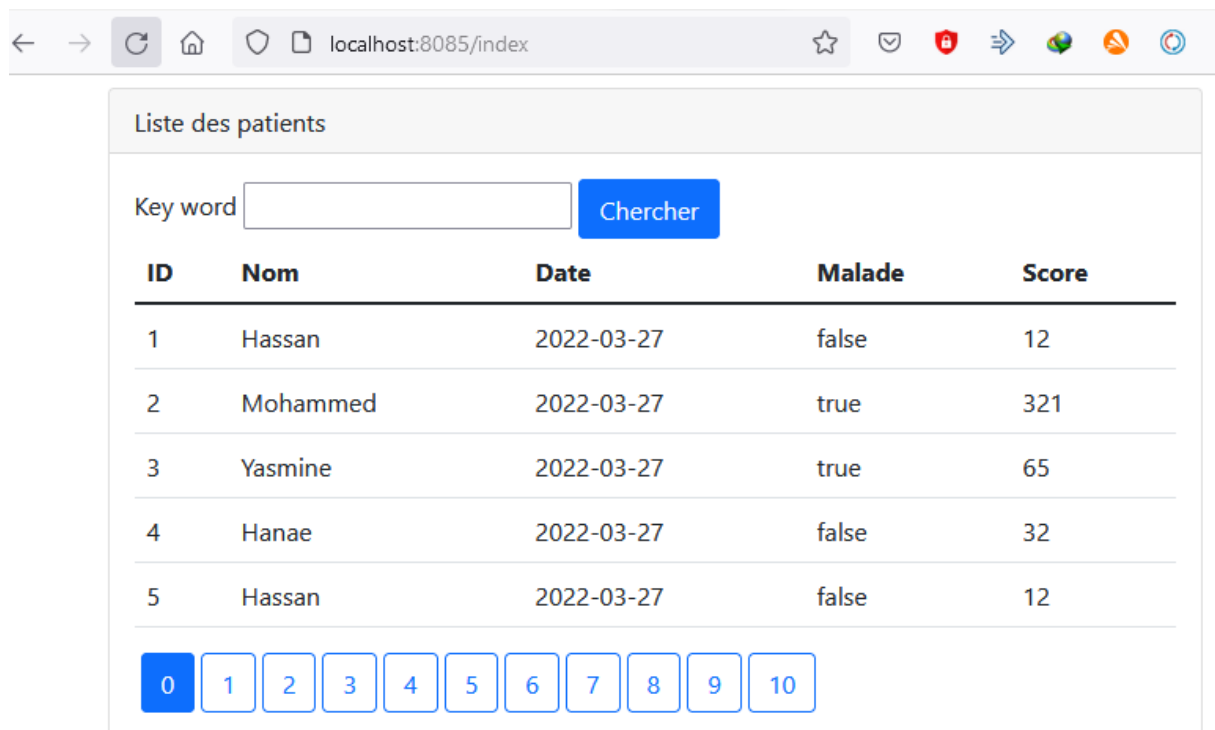
ID	Nom	Date	Malade	Score
21	Hassan	2022-03-27	false	12
22	Mohammed	2022-03-27	true	321
23	Yasmine	2022-03-27	true	65
24	Hanae	2022-03-27	false	32
25	Hassan	2022-03-27	false	12

0 1 2 3 4 5 6 7

Nous allons maintenant passer à la partie de la recherche. L'utilisateur doit pouvoir rechercher un patient avec un mot clé. Pour cela on ajoute dans **patients.html** un formulaire offrant un champ de saisie et un bouton **Chercher** :

```
<div class="card-body">
  <form method="get" th:action="@{index}">
    <label>Key word</label>
    <input type="text" name="keyword">
    <button type="submit" class="btn btn-primary">Chercher</button>
  </form>
  <table class="table">
```

Quand on exécute :



ID	Nom	Date	Malade	Score
1	Hassan	2022-03-27	false	12
2	Mohammed	2022-03-27	true	321
3	Yasmine	2022-03-27	true	65
4	Hanae	2022-03-27	false	32
5	Hassan	2022-03-27	false	12

0 1 2 3 4 5 6 7 8 9 10

On ajoute keyword comme paramètre de l'url et un traitement se fera avec ce keyword. En effet on va prendre le keyword et l'utiliser dans une méthode de **PatientRepository** qui va nous permettre de faire la recherche. Elle va retourner tous les patients dont les noms contiennent keyword.

On aura :

```

@Controller
@AllArgsConstructor
public class PatientController {
    private PatientRepository patientRepository;
    @GetMapping(path = "/index")
    public String patients(Model model,
        @RequestParam(name = "page",defaultValue = "0") int page,
        @RequestParam(name = "size",defaultValue = "5") int size,
        @RequestParam(name = "keyword",defaultValue = "") String keyword){
        Page<Patient> pagePatients = patientRepository.findByNomContains(keyword, PageRequest.of(page,size));
        model.addAttribute( attributeName: "listPatients",pagePatients.getContent());
        model.addAttribute( attributeName: "pages",new int[pagePatients.getTotalPages()]);
        model.addAttribute( attributeName: "currentPage",page);
        return "patients";
    }
}

```

```

public interface PatientRepository extends JpaRepository<Patient,Long> {
    Page<Patient> findByNomContains(String kw, Pageable pageable);
}

```

Quand on exécute et recherche **Mo**, comme on peut le constater dans l'url, on nous affiche la liste des patients dont les noms contiennent **Mo** :

localhost:8085/index?keyword=Mo

Liste des patients

Key word  Chercher

ID	Nom	Date	Malade	Score
2	Mohammed	2022-03-27	true	321
6	Mohammed	2022-03-27	true	321
10	Mohammed	2022-03-27	true	321
14	Mohammed	2022-03-27	true	321
18	Mohammed	2022-03-27	true	321

0 1 2

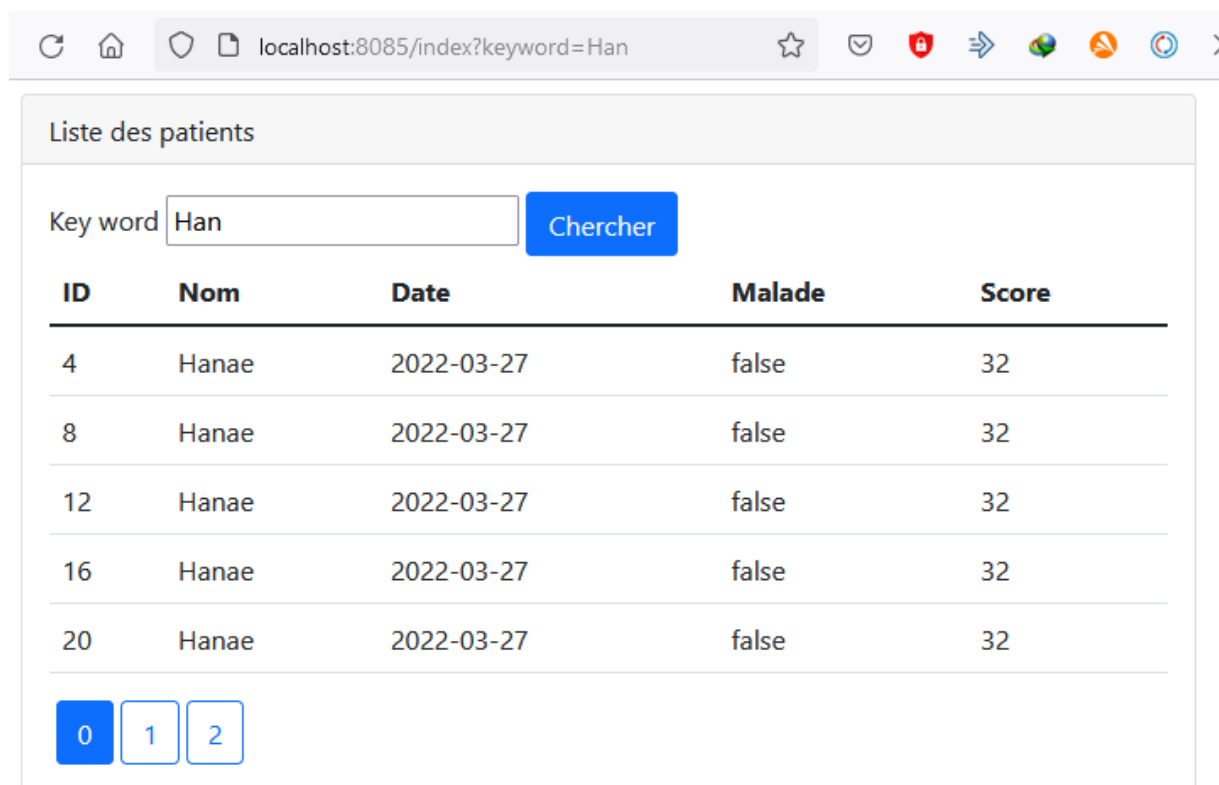
On remarque tout de même que le champ de saisie devient vide. Pour y remédier on peut ajouter un attribut dans le modèle, attribut dans lequel on va stocker keyword.

```
model.addAttribute( attributeName: "currentPage",page);  
model.addAttribute( attributeName: "keyword",keyword);  
return "patients";
```

Puis on donne keyword comme valeur à l'input, le champ de saisie :

```
<input type="text" name="keyword" th:value="${keyword}">  
<button type="submit" class="btn btn-primary">Chercher</button>
```

On peut voir maintenant que quand on fait une recherche, le mot clé ne disparaît pas :



localhost:8085/index?keyword=Han

Liste des patients

Key word

ID	Nom	Date	Malade	Score
4	Hanae	2022-03-27	false	32
8	Hanae	2022-03-27	false	32
12	Hanae	2022-03-27	false	32
16	Hanae	2022-03-27	false	32
20	Hanae	2022-03-27	false	32

0 1 2

Mais on a toujours un problème. Quand par exemple ci-dessus, on clique sur 2 pour afficher la page 2, on perd le mot clé et on obtient l'affichage de la page 2 correspondant à tous les patients. On ne tient donc plus compte de la recherche.

Pour y remédier, sur les liens des numéros de page, on ajoute le keyword. Comme suit :



```

<ul class="nav nav-pills">
  <li th:each="page,status:${pages}">
    <a th:class="${status.index==currentPage?'btn btn-primary ms-1':'btn btn-outline-primary ms-1'}"
      th:text="${status.index}"
      th:href="@{index(page=${status.index},keyword=${keyword})}"
    ></a>
  </li>
</ul>

```

Maintenant quand on fait la recherche on peut parcourir les pages concernant la recherche, les pages correspondant au mot clé saisi, le keyword.

The screenshot shows a web browser at localhost:8085/index?page=2&keyword=Has. The page title is 'Liste des patients'. Below the title is a search bar with the text 'Key word' and a text input field containing 'Has', followed by a blue 'Chercher' button. Below the search bar is a table with 5 columns: ID, Nom, Date, Malade, and Score. The table contains 5 rows of data, all with 'Hassan' as the name and '2022-03-27' as the date. Below the table is a pagination bar with buttons for pages 0, 1, 2, and 3. Page 2 is currently selected.

ID	Nom	Date	Malade	Score
41	Hassan	2022-03-27	false	12
45	Hassan	2022-03-27	false	12
49	Hassan	2022-03-27	false	12
53	Hassan	2022-03-27	false	12
57	Hassan	2022-03-27	false	12

Maintenant passons à la suppression.

Commençons par ajouter une colonne après la colonne score. La colonne contiendra pour chaque patient un bouton **Delete** sur lequel on clique pour supprimer le patient :

```

<td th:text="${p.malade}"></td>
<td th:text="${p.score}"></td>
<td>
  <a class="btn btn-danger" th:href="@{delete(id=${p.id}, keyword=${keyword}, page=${currentPage})}"
    Delete
  </a>
</td>

```

Par la suite on ajoute dans un contrôleur une fonction **delete** qui va être exécutée sur l'adresse **localhost:8085/delete** et qui va faire une redirection.

La fonction **delete** qui prend **id** parmi ses paramètres, supprime le patient ayant cet **id** et la redirection se fait aussi avec les autres paramètres comme **page** et **keyword** de sorte qu'après la suppression on reste sur la même page :

```
@GetMapping("/delete")
public String delete(Long id, String keyword, int page){
    patientRepository.deleteById(id);
    return "redirect:/index?page="+page+"&keyword="+keyword ;
}
```

On va supprimer le patient qui le 14 :

→ ↻ 🏠 🛡️ 📄 localhost:8085/index?keyword=Moh ☆ 📧 🔒 ➡️ 🌐 📱 🔄 >

Liste des patients

Key word

ID	Nom	Date	Malade	Score	
2	Mohammed	2022-03-27	true	321	<input type="button" value="Delete"/>
6	Mohammed	2022-03-27	true	321	<input type="button" value="Delete"/>
10	Mohammed	2022-03-27	true	321	<input type="button" value="Delete"/>
14	Mohammed	2022-03-27	true	321	<input type="button" value="Delete"/>
18	Mohammed	2022-03-27	true	321	<input type="button" value="Delete"/>

Après la suppression on reste sur la même page comme on peut le voir ci-dessous :

localhost:8085/index?page=0&keyword=Moh

### Liste des patients

Key word  [Chercher](#)

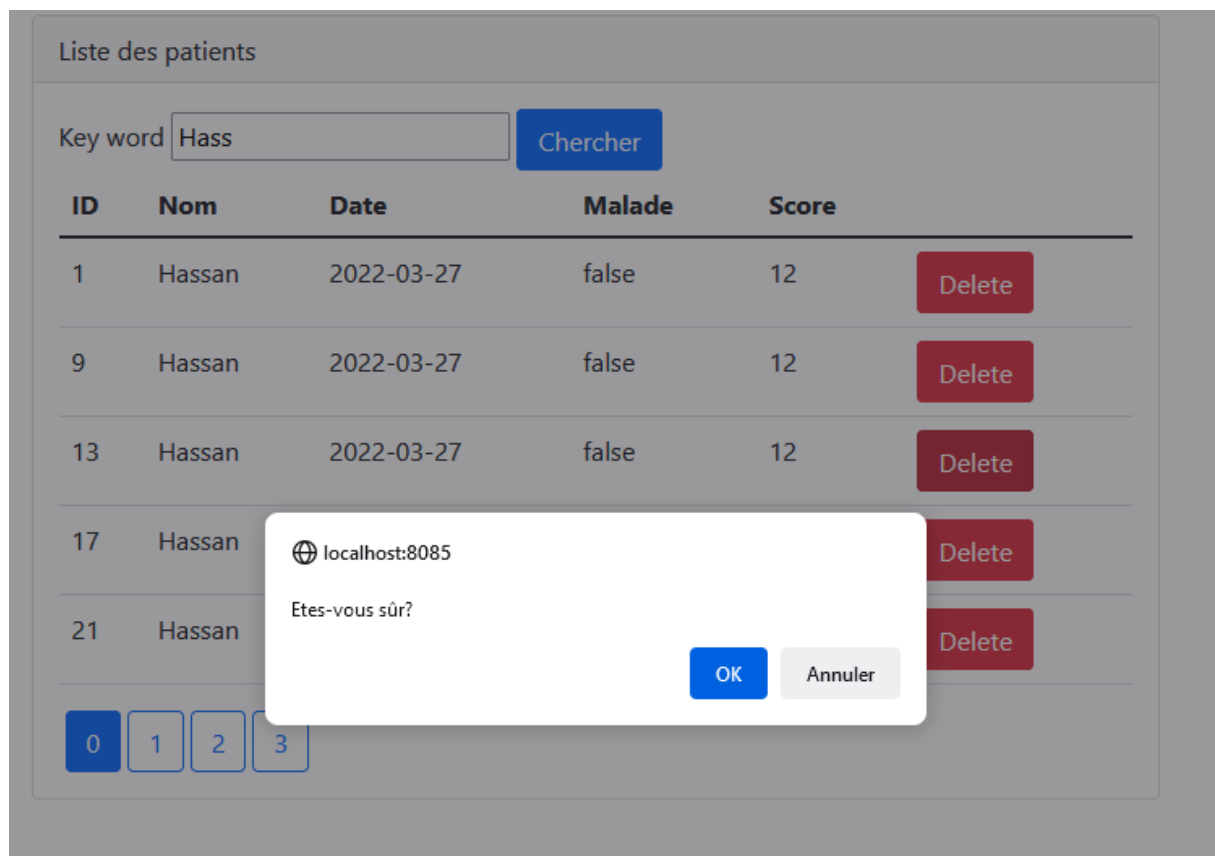
ID	Nom	Date	Malade	Score	
2	Mohammed	2022-03-27	true	321	<a href="#">Delete</a>
6	Mohammed	2022-03-27	true	321	<a href="#">Delete</a>
10	Mohammed	2022-03-27	true	321	<a href="#">Delete</a>
18	Mohammed	2022-03-27	true	321	<a href="#">Delete</a>
22	Mohammed	2022-03-27	true	321	<a href="#">Delete</a>

[0](#)
[1](#)
[2](#)
[3](#)

On va demander la confirmation à l'utilisateur avant la suppression. On fait donc :

```
<td th:text="${p.score}"></td>
<td>
  <a onclick="return confirm('Etes-vous sûr?')" class="btn btn-danger" th:href="@{delete(id=${p.id}, keyword=${keyword}, page=${currentPage})}">
    Delete
  </a>
</td>
```

Maintenant quand on essaie de supprimer, une confirmation est demandée. Quand on fait annuler la suppression ne s'effectue pas. Si on clique sur OK le patient est supprimé :



On clique sur OK et on remarque que le patient qui a id = 13 est supprimé :

localhost:8085/index?page=0&keyword=Hass

### Liste des patients

Key word  Chercher

ID	Nom	Date	Malade	Score	
1	Hassan	2022-03-27	false	12	<button>Delete</button>
9	Hassan	2022-03-27	false	12	<button>Delete</button>
17	Hassan	2022-03-27	false	12	<button>Delete</button>
21	Hassan	2022-03-27	false	12	<button>Delete</button>
25	Hassan	2022-03-27	false	12	<button>Delete</button>

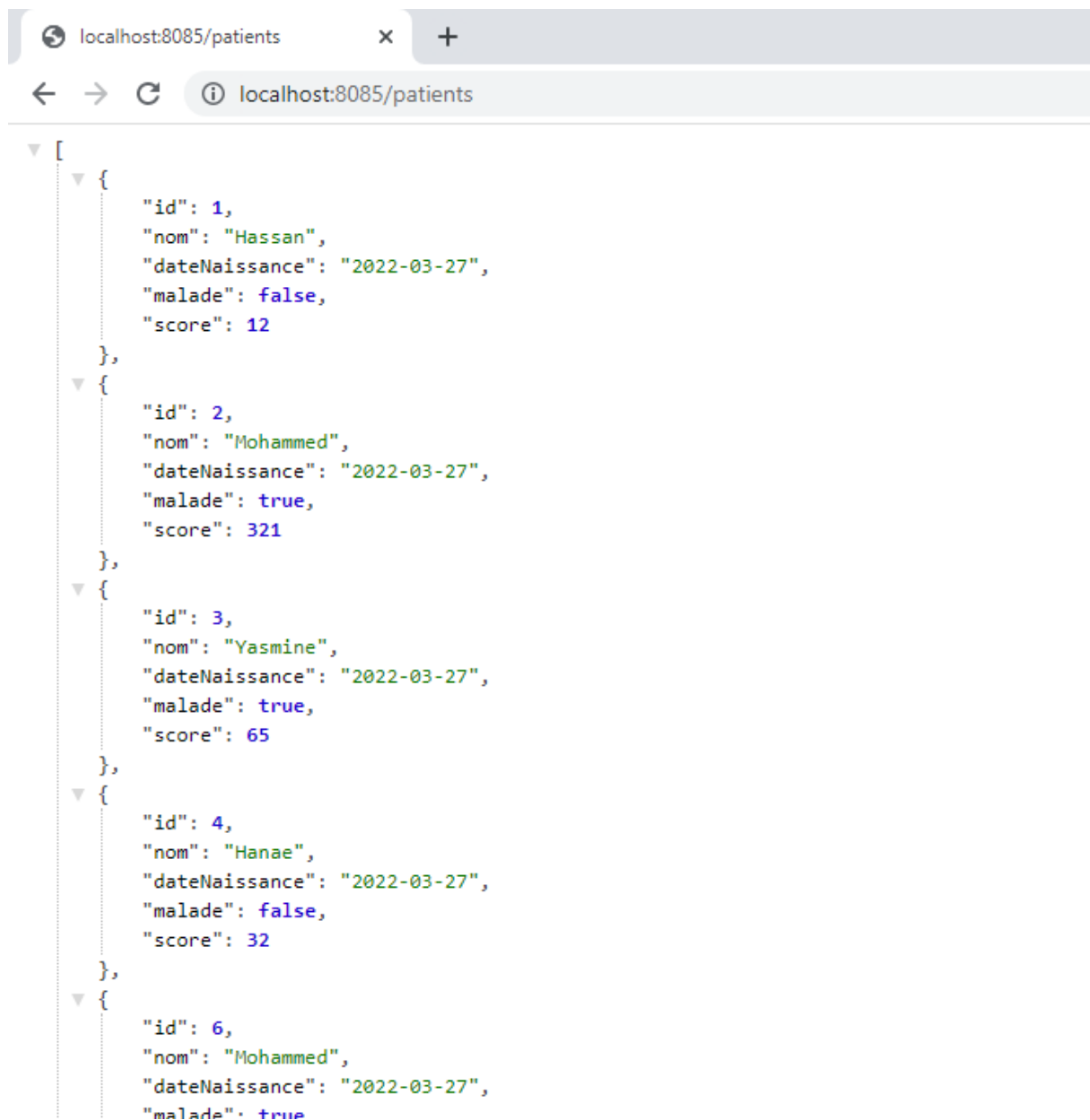
0 1 2 3

On peut aussi retourner sous format JSON la liste des patients. Ce qui nous sera utile lorsque nous feront le traitement côté client.

```
@GetMapping("/")
public String home(){
    return "redirect:/index";
}

@GetMapping("/patients")
@ResponseBody
public List<Patient> listPatients(){
    return patientRepository.findAll();
}
```

Sur l'adresse **localhost:8085/patients** on a :



```
[
  {
    "id": 1,
    "nom": "Hassan",
    "dateNaissance": "2022-03-27",
    "malade": false,
    "score": 12
  },
  {
    "id": 2,
    "nom": "Mohammed",
    "dateNaissance": "2022-03-27",
    "malade": true,
    "score": 321
  },
  {
    "id": 3,
    "nom": "Yasmine",
    "dateNaissance": "2022-03-27",
    "malade": true,
    "score": 65
  },
  {
    "id": 4,
    "nom": "Hanae",
    "dateNaissance": "2022-03-27",
    "malade": false,
    "score": 32
  },
  {
    "id": 6,
    "nom": "Mohammed",
    "dateNaissance": "2022-03-27",
    "malade": true
  }
]
```